Analiza algoritma

Marko Carić

Osnovne operacije

- Dodela vrednosti promenljivoj
- Poređenje dve promenljive
- Aritmetičke operacije
- Logičke operacije
- Ulazno/izlazne operacije

Niz

- Konačna **sekvenca** odnosno **konačni niz** je funkcija iz skupa {1,2,3,4,...,n} u neki skup S. Beskonačna sekvenca odnosno **beskonačni niz** je funkcija iz skupa {1,2,3,4,...} u neki skup S. Sekvencama odnosno nizovima nazivamo i konačne i beskonačne nizove, pri čemu se termin sekvenca češće koristi za označavanje konačnih nizova.
- U velikom broju slučajeva, skup S predstavlja skup prirodnih, celih, racionalnih ili realnih brojeva.

Niz brojeva *A*

- $A(1),\ldots,A(i),\ldots,A(n)$
- i – indeks (pozicija) broja u nizu
- n – veličina niza
- Podniz $A(1,..i)$ veličine i
- Na primer u nizu: 3 5 2 7 1 9 7 $A(3)=2$, $n=7$
- Primer rastućeg niza: 1 3 4 6 7 8 9
- Neka je $A(i)=i+5 \Rightarrow$
  $A(1)=1+5=6,\ldots,A(5)=5+5=10,\ldots, A(n)=n+5$

Suma

- Suma $A_r+A_{r+1}+A_{r+2}+A_{r+3}+\ldots+A_{r+k}$ može se zapisati sumacionom notacijom u obliku:

$$\sum_{i=r}^{r+k} A_i$$

- Na primer, suma prvih n prirodnih brojeva je

$$\sum_{i=1}^{n} i = 1+2+\ldots+n = \frac{n(n+1)}{2}$$

Broj iteracija pri najgorem slučaju izvršavanja

- n = 5;
- **Ponavljaj**
  - **Učitaj** (m);
  - n = n -1;
- **Sve dok ne bude** (m==0) ili (n==0)

Prevedena sintaksa

- n = 5;
- **Repeat**
  - **Read**(m);
  - n = n -1;
- **Until** (m==0) ∨ (n==0)

Klasična sintaksa

T(n) – vreme izvršavanja u najgorem slučaju

- **for** i=1 **to** n **do**
  - **for** j=1 **to** n **do**
    - **if**(i<j) **then**
      - **swap**(A(i,j),A(j,i))

- swap (zamena) se posmatra kao osnovna operacija
- $T(n) = n \cdot n \cdot 2 = 2 \cdot n^2$
- Može se posmatrati i *prosečno* vreme izvršavanja
- Kod randomiziranih algoritama (sa slučajnim odlukama) posmatra se *očekivano* vreme izvršavanja

Zamena vrednosti dve promenljive – konstantno vreme izvršavanja

- Swap (x,y)
  - pom = x;
  - x = y;
  - y = pom;
  - **return** x,y;

- T = 1 + 1 + 1 + 1 = 4
- Za algoritam swap se kaže da se izvršava u *konstantnom* vremenu.

Maksimalni element niza – linearno vreme izvršavanja

- PronađiNajvećiElement
  - m = A(1);
  - j = 1;
  - i = 2;
  - **while** i$\leq$n **do**
    - **if** m<A(i) **then**
      - m = A(i);
      - j = i;
    - i = i+1;
  - **return** j;

  - T(n) = 1+1+1+(n−1)(3+2)+1=5n−1
  - Algoritam PNE se izvršava u *linearnom* vremenu.

Sortiranje niza zamenjivanjem (bubble-sort) – kvadratno vreme

- Bubble-Sort(*A*)
  - **for** i=1 **to** n-1 **do**
    - **for** j=n **downto** i+1 **do**
      - **if**(*A*(j)<*A*(j-1) **then**
        - swap(*A*(j),*A*(j-1));
  - **return** *A*;

$$T(n) = 2 + \sum_{i=1}^{n-1} (3 + 4(n-i)) =$$

$$2 + \sum_{i=1}^{n-1} 3 + 4\sum_{i=1}^{n-1} (n-i) = 2 + 3(n-1) + 4\sum_{i=1}^{n-1} i$$

$$= 2 + 3n - 3 + 4\frac{(n-1)n}{2} = 2n^2 + n - 1$$

Kvadratno vreme izvršavanja

- Algoritam bubble-sort ima *kvadratno* vreme izvršavanja ($2n^2+n+1$)
- U najboljem slučaju kada je ulazni niz *A* početno već sortiran, zamena mesta susednih elemenata se ne dešava, ali algoritam bubble-sort ipak zahteva slično vreme izvršavanja pošto se proveravanje uslova **if** obavlja $\frac{n(n-1)}{2}$ puta.

Pristupi za izračunavanje $f(n)=a^n$

- Neka je $P=1$;
- Učitaj(a,n);
- **for** $i=1$ **to** n
  - Zamenimo vrednost P vrednošću $P \cdot a$;
- Izlaz: P

Ne rekurzivno rešenje

- $f(0)=1$
- $f(k+1)=a \cdot f(k)$

Rekurzivno rešenje

Računanje faktorijala – n!

- Fakt(n)
- **if**(n==0) **then** Fakt=1;
- Neka je Fakt=1;
- **for** k=1 **to** n
  - Vrednost Fakt zameni vrednošču k · Fakt
- **return** Fakt;

Ne rekurzivno

- Fakt(n)
- **if**(n==0) **then**
  Fakt=1;
- **if**(n>0) **then**
  Fakt=n*Fakt(n-1);
- **return** Fakt;

Rekurzivno

Binarna pretraga (sortiranog niza) – nerekurzivno rešenje

- Bin–Search(A,n,x)
- i=1;j=n;
- **while** (i≤j) **do**
  - k = (i+j)/2;
  - **if**(x<A[k]) **then**
    - j = k–1;
  - **else if** (x>A[k]) **then**
    - i = k+1;
  - **else**
    - **return** k;
- **return** 0;

Sekvencijalna pretraga sortiranog niza

- Seq-Search(A,x)
  - j=0;
  - i=1;
- **while** (i≤n ∧ j==0) **do**
  - if(A(i)==x) **then** j=i;
  - i=i+1;
- **return** j;

- Crvene naredbe se dešavaju najviše jednom
- Plave naredbe se dešavaju najviše n puta
- T(n)=4n+4 (u najgorem slučaju)

Binarna pretraga sortiranog niza – logaritamsko vreme izvršavanja

- Bin–Search(*A*,i,j,x)
  - **if** j<i **then**
    - **return** 0;   // niz je prazan
  - m = $\lfloor (i+j)/2 \rfloor$;
  - **if** *A*(m)==x **then**
    - **return** m;
  - **else if** *A*(m)>x then
    - **return** Bin–Search(*A*,i,m–1,x);
  - **else**
    - **return** Bin–Search(*A*,m+1,j,x);

Logaritamsko vreme izvršavanja

- $T(1)=c$
- $T(n)=T(n/2)+d=T(n/4)+d+d=T(n/4)+2d$

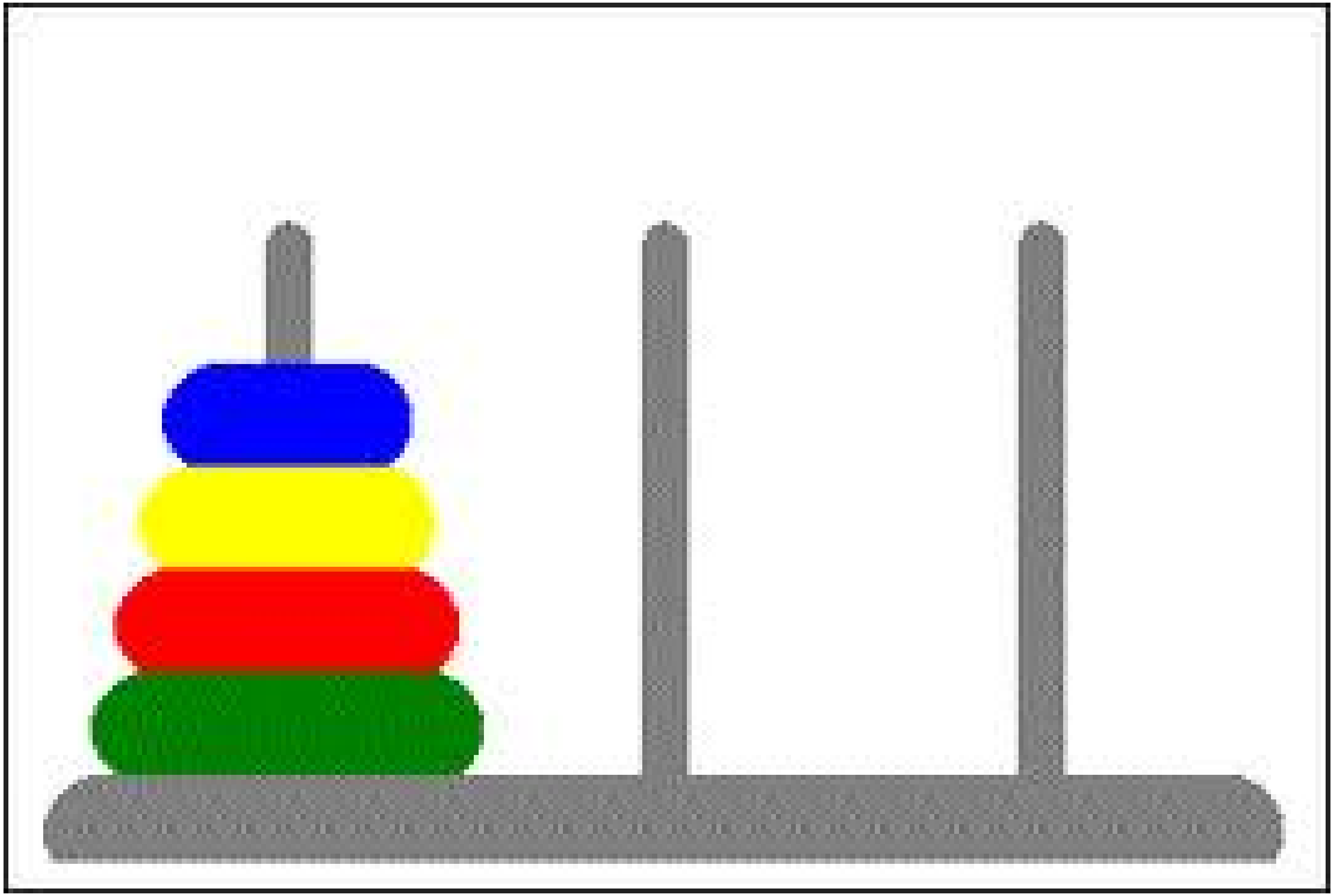
$=T(n/8)+d+2d=T(n/8)+3d=...=T(n/2^k)+kd$

- Ako pretpostavimo da je $n=2^k$ za $k\geq 1$ tj. $k=\log_2(n)$ sledi:

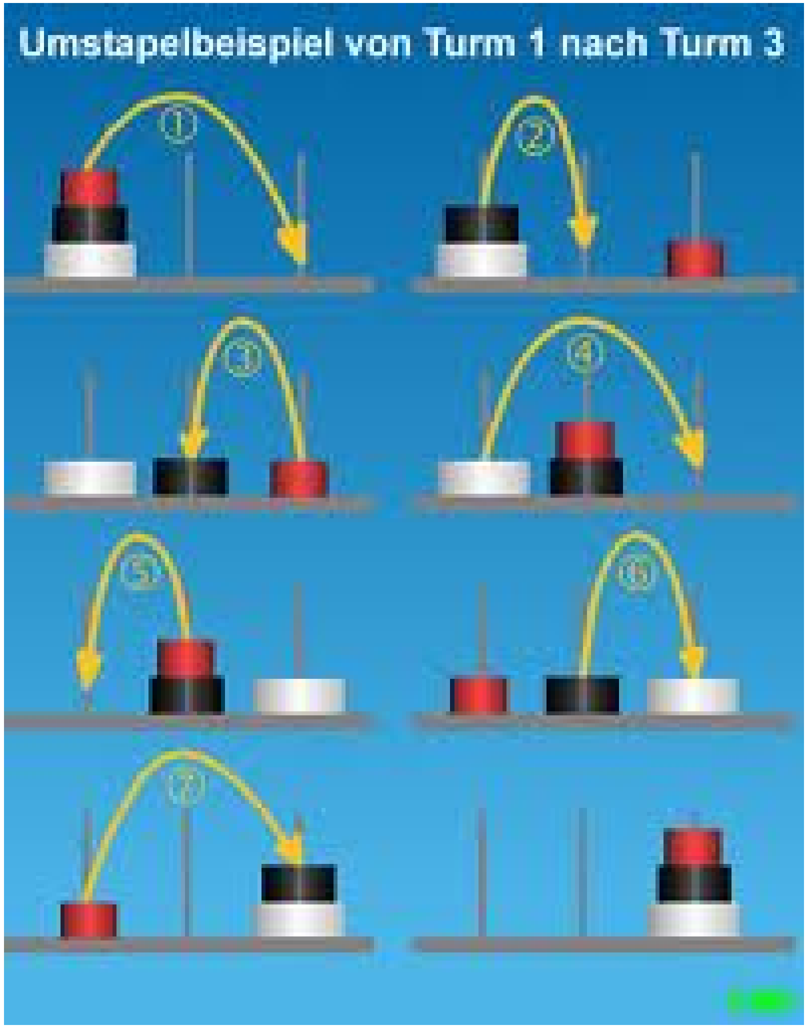
  $T(n/2^k)+kd=T(1)+\log_2(n)\cdot d=c+d\cdot\log_2(n)$

- T(1) označava konstantno vreme izvršavanja
- Kaže se da binarna pretraga ima *logaritamsko* vreme izvršavanja.

Problem Hanojskih kula



Problem Hanojskih kula slučaj n=3



Hanojske kule – rekurzivno rešenje

- podprogram
  hanojskekule(n, a, b, c)
- **if**(n>0)
- hanojskekule(n-1,a,c,b);
- Ispiši ("Prebaci sa "+a+" na "+b);
- hanojskekule(n-1,c,b,a);
-
- glavni program
- n=3; a=1; b=3; c=2;
- hanojskekule(n,a,b,c);

Problem hanojskih kula – eksponencijalno vreme izvršavanja

- Neka je $H(k)$ broj pokreta potrebnih da se k diskova prebaci sa štapa 1 na štap 3
- Pretpostavimo da je poznata vrednost $H(k)$; tada je $H(k+1) = H(k)+1+H(k)=2\cdot H(k)+1$
- $H(1)=1$ $\quad H(2)= 2\cdot 1+1=3$
- $H(3)=2\cdot 3+1=7$ $\quad H(4)= 2\cdot 7+1=15=2^4-1$
- Naslućuje se da je $H(k)=2^k-1$
- $H(k+1)=2\cdot H(k)+1=2\cdot (2^k-1)+1=2^{k+1}-1$
- Kažemo da problem Hanojskih kula ima *eksponencijalno* vreme izvršavanja.

Brzina rasta funkcija

- $f_1(n) = 10 \cdot n^3 + n^2 + 40 \cdot n + 80$
- $f_2(n) = 17 \cdot n \cdot \log(n) - 23 \cdot n - 10$
- $g_1(n) = n^3$
- $g_2(n) = n \cdot \log(n)$

- Za velike vrednosti promenljive n važi:
- $f_1(n) \approx g_1(n)$
- $f_2(n) \approx g_2(n)$

- Za dovoljno veliko n vrednost funkcije je određena njenim dominantnim izrazom.

Tipične funkcije poređane po rastućim brzinama rasta

Funkcija	Ime
1	konstantna funkcija
log(n)	logaritamska funkcija
n	linearna funkcija
$n \cdot \log(n)$	$n \cdot \log(n)$
n^2	kvadratna funkcija
n^3	kubna funkcija
2^n	eksponencijalna funkcija

Broj operacija i njihovo trajanje – računar obavlja milion operacija u sekundi

n (ulaz)	$A1=2^n$	$A2=5n^2$	$A3=100n$
1	2	5	100
10	1024	500	1000
100	2^{100}	50000	10000
1000	2^{1000}	$5 \cdot 10^6$	100000

n	$A1=2^n$	$A2=5n^2$	$A3=100n$
1	1µs	5 µs	100 µs
10	1 ms	0,5 ms	1 ms
100	2^{70} god	0,05 s	0,01 s
1000	2^{970} god	5 s	0,1 s

Trajanje algoritma u sekundama – računar obavlja m operacija u sekundi za ulaz n=1000

m\Alg	$\log_2 n$	n	$n \cdot \log_2 n$	$n^{1.5}$	n^2	n^3	1.1^n
10^3	0.01	1	10	32.0	1000	10^6	10^{39}
10^4	0.001	0.1	1	3.2	100	10^5	10^{38}
10^5	0.0001	0.01	0.1	0.32	10	10^4	10^{37}
10^6	0.00001	0.001	0.01	0.032	1	10^3	10^{36}

Asimptotska notacija

- **Definicija (Veliko O)**: Za dve nenegativne funkcije $f,g:N\to R$ kažemo da je $f(n)=O(g(n))$ ako postoje pozitivne konstante c i n_0 tako da je $f(n)\leq c\cdot g(n)$ za svako $n\geq n_0$.
- Na primer, pokažimo da je izvršavanje algoritma Bubble-sort jednako $O(n^2)$:

  $2n^2+n-1\leq 2n^2+n\leq 2n^2+n^2=3n^2$
- Dakle, izborom $n_0=1$ i $c=3$ sledi $T(n)=O(n^2)$
- Alternativno, može se izabrati i $c=2.2$ i $n_0=4$, pošto je $2n^2+n-1\leq 2.2\cdot n^2$.

Asimptotska notacija (2)

- U stvari, možemo tvrditi i da je $T(n)=2n^2+n-1$ veliko O od svakog količnika vrednosti n^2, recimo od $O(n^2/100)$.
- Potrebno je izabrati $n_0=1$ i $c=300$;
- Tada je $2n^2+n-1 \leq 2n^2+n \leq 2n^2+n^2=3n^2= 300 \cdot n^2/100$, tj. $T(n)=O(n^2/100)$.
- Iz prethodnog sledi da konstantni faktori nisu važni, tj. za svaku pozitivnu konstantu d i za svaku funkciju $T(n)$, važi $T(n)=O(d \cdot T(n))$.

Asimptotska notacija (3)

- **Definicija (Ω–zapis)** Za dve nenegativne funkcije $f,g:N \to R^+$ kažemo da je $f(n)= \Omega(g(n))$ ako postoje pozitivne konstante c i n_0 tako da je $f(n) \geq c \cdot g(n)$ za svako $n \geq n_0$.
- **Definicija (Θ–zapis)** Za dve nenegativne funkcije $f,g:N \to R^+$ kažemo da je $f(n)= \Theta(g(n))$ ako je $f(n)=O(g(n))$ i $f(n)= \Omega(g(n))$.
- **Definicija (malo o)**: Za dve nenegativne funkcije $f,g:N \to R$ kažemo da je $f(n)=o(g(n))$ ako za svaku pozitivnu konstantu c postoji konstanta n_0 tako da je $f(n) \leq c \cdot g(n)$ za svako $n \geq n_0$.

Asimptotska notacija (4)

Asimptotska notacija	Intuitivno značenje
$f(n)=O(g(n))$	$f \leq g$
$f(n)=\Omega(g(n))$	$f \geq g$
$f(n)=\Theta(g(n))$	$f \approx g$
$f(n)=o(g(n))$	$f \ll g$

Primer

- Pokažimo da je $n^3+n^2-1=\Theta(n^3)$.
- Izaberimo pozitivne konstante c_1, c_2 i n_0 tako da je $c_1 n^3 \leq n^3+n^2-1 \leq c_2 n^3$, za svako $n \geq n_0$.
- Deljenjem ovih nejednakosti sa n^3 dobijamo:

$$c_1 \leq 1+\frac{1}{n}-\frac{1}{n^2} \leq c_2$$

- Sada se vidi da se za konstante može uzeti $c_1=1$, $c_2=2$ i $n_0=1$.

Sortiranje umetanjem – kvadratno vreme izvršavanja

- Insert-Sort(A)
- $A(0)=-\infty$;
- **for** i=2 **to** n **do**
  - j=i;
  - **while** (A(j)<A(j-1)) **do**
    - swap(A(j),A(j-1));
    - j = j-1;
- **return** A;

$$T(n)=c_1+\sum_{i=2}^{n}(c_2+c_3(i-1))$$

$$=c_1+c_2(n-1)+c_3\sum_{i=2}^{n}(i-1)$$

$$=c_1+c_2(n-1)+c_3\frac{(n-1)n}{2}=c_4(n-1)n$$

Sortiranje izborom – kvadratno vreme izvršavanja

- Select-Sort(A)
- **for** i=1 **to** n-1 **do**
  - j = FindMin(A,i,n);
  - Swap(A(i),A(j));
- **return** A;

$$T(n) = c_3 + \sum_{i=1}^{n-1}(c_2 + c_1(n-i))$$

$$= c_3 + c_2(n-1) + c_3\sum_{i=2}^{n}(i-1)$$

$$= c_3 + c_2(n-1) + c_1\frac{(n-1)n}{2} \leq c_4(n-1)n$$

Algoritam razdvajanja

- Razdvajanje(A,Levi,Desni)
- Ulaz: A (niz), Levi (leva granica niza) i Desni (desna granica niza).
- Izlaz: A i S = Razdvajanje, takav indeks da je A[i]≤A[S] za sve i≤S i A[j] > A[S] za sve j > S.
  - pivot = A[Levi];
  - L = Levi; D = Desni;
  - **while** (L < D) **do**
  - **while** (A[L]≤pivot **and** L≤Desni) **do** L = L + 1;
  - **while** (A[D] > pivot **and** D≥Levi) **do** D = D – 1;
  - **if** (L < D) **then**
    - zameni A[L] i A[D];
  - Razdvajanje = D;
  - zameni A[Levi] sa A[D];

Razdvajanje elementa oko pivota 6

- 6 2 8 5 10 9 12 1 15 7 3 13 4 11 16 14
- 6 2 **4** 5 10 9 12 1 15 7 3 13 **8** 11 16 14
- 6 2 4 5 **3** 9 12 1 15 7 **10** 13 8 11 16 14
- 6 2 4 5 3 **1** 12 **9** 15 7 10 13 8 11 16 14
- **1** 2 4 5 3 **6** 12 9 15 7 10 13 8 11 16 14

Brzo sortiranje (razdvajanjem)

- Quicksort(A, n)
- Ulaz: A (niz od n brojeva).
- Izlaz: A (sortirani niz).
- Početak glavnog programa
  - Q_sort(1,n)
- Kraj glavnog programa
- podprogram Q_Sort(Levi,Desni);
  - **if** Levi < Desni **then**
    - S = Razdvajanje(A,Levi,Desni);
  - //S je indeks pivota posle razdvajanja
  - Q_Sort(Levi,S – 1);
  - Q_Sort(S + 1,Desni);

Brzo sortiranje primer

- 6 2 8 5 10 9 12 1 15 7 3 13 4 11 16 14
- 1 2 4 5 3 **6** 12 9 15 7 10 13 8 11 16 14
- **1** 2 4 5 3

  2 4 5 3

  3 **4** 5

  8 9 11 7 10 **12** 13 15 16 14

  7 **8** 11 9 10

  10 9 **11**

  9 **10**

  13 15 16 14

  14 **15** 16
- 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Brzo sortiranje – vreme izvršavanja

- $T(n) = O(n \cdot \log n)$
- Brzo sortiranje (Quick-sort) je dakle brz u proseku, iako je u najgorem slučaju vreme njegovog izvršavanja kvadratno.